Using dynamic partial reconfiguration of FPGAs in real-Time systems

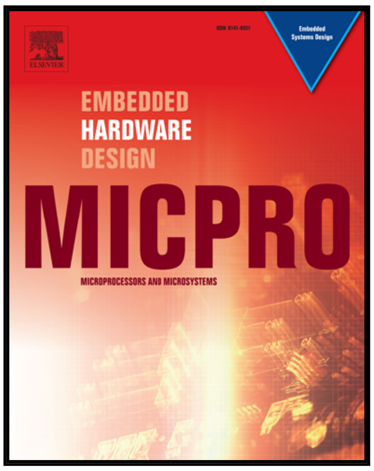
Pezzarossa, Luca; Kristensen, Andreas Toftegaard; Schoeberl, Martin; Sparsø, Jens

Published in:
Microprocessors and Microsystems

Link to article, DOI:
10.1016/j.micpro.2018.05.017

Publication date:
2018

Document Version
Peer reviewed version



Using Dynamic Partial Reconfiguration of FPGAs in Real-Time Systems

Luca Pezzarossa, Andreas Toftegaard Kristensen, Martin Schoeberl, Jens Sparsø

PII:	S0141-9331(18)30080-2
DOI:	10.1016/j.micpro.2018.05.017
Reference:	MICPRO 2698

To appear in: *Microprocessors and Microsystems*

Received date:	16 February 2018
Accepted date:	25 May 2018

Using Dynamic Partial Reconfiguration of FPGAs in Real-Time Systems

Luca Pezzarossa*, Andreas Toftegaard Kristensen, Martin Schoeberl, and Jens Sparsø

Department of Applied Mathematics and Computer Science
Technical University of Denmark
Kongens Lyngby, Denmark

Abstract

The use of hardware accelerators to implement computationally intensive tasks in real-time systems can lead to a reduction of the worst-case execution time (WCET). An additional potential benefit is that a WCET-analysis may be simpler to perform because hardware generally has a more time-predictable behavior than software. The dynamic partial reconfiguration (DPR) feature offered by modern FPGAs allows accelerators that are no longer needed to be replaced with new ones, leading to more efficient utilization of hardware resources. This paper presents an experimental evaluation of the potential benefits of using DPR to implement hardware accelerators in real-time systems, focusing on trade-offs between hardware utilization, worst-case performance, and speed-up over a pure software solution. Moreover, it also investigates the trade-off between the use of multiple specialized accelerators combined with DPR instead of the use of a more general accelerator, and the memory footprint of the partial-bit streams. The experiments show that DPR in combination with accelerators results in: (i) better utilization of the FPGA resources, (ii) performance that is comparable with non-reconfigurable solutions, and (iii) tighter WCET bounds.

1. Introduction

In recent years, advances in FPGA technology have enabled reconfigurable computing to become viable and be used in end-products. This empowers application developers to design and use their own hardware accelerators (HwAs) to significantly increase the speed of algorithms by using reconfigurable hardware.

In real-time systems, tasks need to meet their deadlines. To guarantee that no deadlines are missed, the worst-case execution time (WCET) of tasks needs to be determined statically. Moving functionality from software into hardware can lead to a reduction of the WCET. Furthermore, timing-analysis of algorithms in hardware may be easier to perform than WCET analysis of software solutions.

However, one of the issues in using FPGAs is that the size of the HwAs is limited by the available resources, especially considering that the FPGA cost is very sensitive to its size. The dynamic partial reconfiguration (DPR) feature offered by modern FPGAs [1] can be used to overcome this limitation, by enabling run-time reconfiguration of selected regions on the FPGA. This allows a more efficient utilization of FPGA resources since HwAs that are only required for limited amounts of time can be replaced when the functionality implemented in these regions is no longer required.

In this paper, we explore and evaluate the use of DPR in real-time systems. The idea is to equip a processor with

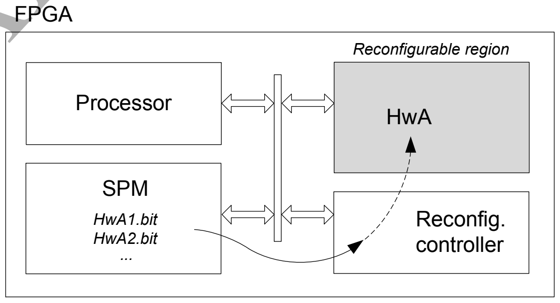


Figure 1: An FPGA with partial reconfiguration capabilities used to instantiate different hardware accelerators during runtime.

one or more reconfigurable regions implementing hardware to accelerate selected segments of a software application, as shown in Figure 1. The DPR feature of FPGAs is used for swapping between these accelerators according to the current needs of the application. The reconfiguration controller modifies the hardware in the reconfigurable region by loading partial bit-streams stored in a local scratchpad memory (SPM) (e.g., *HwA1.bit*, *HwA2.bit* in Figure 1).

Potential outcomes of using reconfiguration in real-time systems are:

- **Reduction of the hardware size:** Implementing different functionalities that are needed only for a limited period of time in the reconfigurable regions reduces the overall hardware size when compared to a fully static solution. This enables a more extensive use of accelerators and possible reduction of costs.

*Corresponding author
Email address: lpez@dtu.dk (Luca Pezzarossa)

- **Simplification of the WCET analysis:** Moving the functionality performed in software into hardware may lead to a simplification of the WCET analysis. In general, the time analysis of hardware used to implement software-equivalent tasks is often easier to perform than analysis of a pure software solution. For example, instruction cache related timing analysis is not needed when using accelerators.

- **Reduction of the WCET pessimism:** An interesting consequence of the simplification of the WCET analysis when using hardware accelerators to perform selected tasks, is the possibility to reduce WCET analysis pessimism. A properly designed hardware accelerator may have a very limited and predictable variance on the task execution times.

- **Speed-up from hardware acceleration:** In general, executing a computationally intensive task in hardware delivers a speed-up in the task execution time, leading to increased system performance and efficiency. This can apply for both the average-case execution time for general-purpose systems and the WCET for real-time systems. Moreover, DPR may allow the use of more efficient accelerators specialized in the execution of small specific tasks instead of a generic accelerator that covers a broader task set.

- **Increase of the design complexity:** The hardware architecture of a system that includes the DPR feature is more complex. For example, the use of a dedicated configuration controller is required, and some interface logic may be needed to isolate the reconfigurable region from the rest of the system during reconfiguration. We mitigate this negative outcome by using our lightweight configuration controller RT-ICAP, characterised by a minimal hardware cost overhead and easy usability.

- **Increase of the memory requirements:** The partial bit-streams associated to each configuration must be stored in memory. Therefore, we expect an increase in the memory resources utilisation when using DPR. This memory increase goes against the hardware size reduction obtained by sharing the reconfigurable region between multiple accelerators. In our approach, we mitigate this negative outcome by applying compression techniques to the stored partial bit-streams.

In this paper, we experimentally evaluate the above mentioned potential advantages. First, we present the reconfiguration controller that we developed to support time-predictable reconfiguration and the associated software tools for reconfiguration time analysis and bit-stream compression. Then, we present a set of experiments comparing a pure software and a static implementation in which non-reconfigurable HwAs are used, against a reconfigurable approach in which DPR is used to switch between different HwAs. The goal of the experiments is to analyze the trade-offs between the hardware-resource utilization and the computational performance loss due to the reconfiguration time overhead of DPR, which directly affects the overall WCET. For one of the test cases, we also investigate whether using DPR to switch between multiple specialized HwAs could provide a lower WCET bound with respect to the use of a more general HwA. We also present a characterization in terms of hardware resources utilization of the controller and bit-stream compression capabilities of the associated tool.

We carry out the evaluation on the Patmos processor [2, 3] using HwAs generated from four selected real-time TACLe suite benchmarks [4]. We generate the hardware from C code using the Xilinx high-level synthesis (HLS) tool Vivado HLS [5]. We use the Patmos WCET analysis tool *platin* [6] to perform the WCET analysis of the accelerated tasks and the reconfiguration process. Overall, the results show that the use of DPR can lead to a significant reduction in the hardware cost if the tasks moved into hardware are sufficiently computationally intensive.

This paper extends our work presented in [7], where we discussed and evaluated the computational performance vs. hardware utilization trade-offs in terms of WCET, comparing a solution that uses hardware accelerators in combination with DPR against a static solution. We extend the previous work with the following additional contributions: (1) an extended discussion of the potential benefits of DPR in real-time systems, (2) a presentation and characterization of our time-predictable reconfiguration controller, (3) an evaluation of the effect of reconfiguration on the WCET speed-up between a software solution and one that uses hardware accelerators in combination with DPR, and (4) an evaluation of the bit-stream memory footprint and compression capabilities of the tool associated with our controller.

The paper is organized into seven sections: Section 2 presents the related work. Section 3 provides the general background related to DPR, the Patmos processor, and its WCET analysis tool. Section 4 presents the reconfiguration controller and the associated software tools for reconfiguration time analysis. Section 5 describes the experimental setup used to produce the results, which consists of the hardware platform and the set of HwAs from the TACLe benchmark suite. Section 6 presents and discusses the experimental results. Finally, Section 7 concludes the paper.

2. Related Work

The related work falls into two categories that we will discuss below: (1) system and application level methods and tools supporting the use of DPR, and (2) hardware controllers used to install new partial bit-streams in the FPGA at runtime.

2.1. Methods and Tools

The use of DPR in real-time systems represents a novel field of research. Below we first discuss some representative works that address and introduce DPR in general, and then a few works with a specific focus on real-time systems.

Comprehensive surveys of the most relevant hardware aspects and an overview of the state-of-the-art with regards to reconfigurable computing can be found in [8] and [9], respectively. These works explore the challenges of runtime reconfigurable architectures, addressing both single-chip and multi-chip architectures.

In [10], the author studies the dynamic behavior of reconfigurable architectures, primarily focusing on the use of DPR. The work proposes a simulation framework for reconfigurable architectures that comprises a generic application model and an architecture model, the combination of which captures the dynamic behavior of the reconfigurable architectures. The work does not address real-time aspects.

Another framework is the one presented in [11], called ReCoBus-builder, which addresses component-based, reconfigurable, non-real-time systems. It uses DPR to generate dynamically reconfigurable systems providing one or more runtime reconfigurable areas. The framework addresses component-based reconfigurable architecture for non-real-time general-purpose systems. Therefore, strict timing constraints are not taken into account and the performance is measured in terms of the average-case speed-up.

The work presented in [12] provides an overview of the hardware-software partitioning, scheduling, and placement issues and proposes an exact and a heuristic approach for hardware-software partitioning in a system that uses DPR. The paper takes into account key factors such as placement implications and configuration pre-fetching for minimizing the schedule length, but it does not address real-time applications.

In [13], the authors present the PaRA-Sched automated design methodology. It considers DPR in the scheduling infrastructure aiming at improving overall performance by masking reconfiguration time when possible. This allows rapid exploration of the DPR during the early stages of the design process.

The work presented in [14] proposes a software framework, called FRED, which exploits HwAs combined with DPR in the development of safety-critical real-time systems. It presents generic models of the platform and the computational workload where a subset of the tasks is accelerated using reconfigurable hardware. These models allow response-time analysis to verify the schedulability of a real-time task set under given constraints and assumptions. In contrast to this modeling and scheduling work, our paper is more focused on practical design experiments.

2.2. Reconfiguration Controllers

A number of reconfiguration controllers have been developed by FPGA vendors and in academia. Again, most of these works do not consider real-time aspects.

The XPS_HWICAP reconfiguration controller [15], provided by Xilinx, is an IP core that supports DPR using a set of software functions provided in processor-specific libraries. The associated software library, allows an application programmer to write and read configuration bit-streams, and it enables the modification of single look-up tables and flip-flop properties.

The ZyCAP controller [16] is a custom controller for processor/FPGA hybrid platforms, such as the Xilinx Zynq. The ZyCAP controller is connected to the hard-core processor through the AXI4-Lite bus and to the system memory where the bit-streams are stored through the AXI4 bus [17]. A direct memory access controller loads the bit-stream during reconfiguration using the internal configuration access port (ICAP). Software drivers are associated with the controller and allow the hard-core processor to manage the reconfiguration process. Most of the functionalities of the above-mentioned controllers are provided by software executing on a processor that reads from or writes to the controller interface. This approach is very flexible, but it may lead to an increase the WCET pessimism, since I/O functions may be difficult to analyze.

The PRC controller [18], provided by Xilinx, is an IP core designed to manage DPR and targets the Xilinx 7-series FPGAs. The controller is interfaced to a processor through the AXI4-Lite bus [17]. When it receives a software or hardware trigger, it can independently handle the reconfiguration of selected regions by reading bit-streams from a memory connected to the AXI4-Lite bus and writing these into the ICAP interface. For this IP, only the netlist is available making the computation of the reconfiguration time very difficult.

The DPRM controller presented in [19] offers similar functionality for the Xilinx Virtex-5 FPGA, but it only supports bit-stream transfers from off-chip flash memories into the FPGA configuration memory. This controller uses a finite-state machine (FSM) to transfer the partial bit-streams stored in off-chip flash memories into the FPGA configuration memory.

The D^{2}PR controller presented in [20] is a minimal custom DPR controller connected to the ICAP interface, which can be configured to include circuitry for configurable error detection and correction to improve safety in DPR by checking for data errors while loading partial bit-streams.

The architecture of the last two controllers is the most similar to our RT-ICAP controller, presented in Section 4. However, they only offer the transfer of bit-streams from a memory to the ICAP interface, while our controller offers additional functionalities, such as support of bit-stream decompression, and a register-based (*status*/*control*) interface.

3. Background

In this section, we present the general background related to DPR, the Patmos processor, and its WCET analysis tool.

3.1. Dynamic Partial Reconfiguration

DPR is a feature of modern FPGAs that allows runtime modification of an operating FPGA [1]. Partial bit-streams can be loaded into the FPGA to reconfigure selected regions without affecting the functionality of other parts of the device. A system that uses DPR can be conceptually divided into two main parts: a non-reconfigurable static part configured at boot-time with a full bit-stream, and a run-time reconfigurable part, which may consist of many independent reconfigurable regions. Each of these regions can be reconfigured multiple times during run-time with different partial bit-streams without interfering with the functionality of the static part.

For Xilinx FPGAs, DPR is performed by loading a partial bit-stream through one of the FPGA configuration interfaces. For this work, the ICAP (internal configuration access port) is used to provide access to the configuration memory and partially reconfigure the FPGA after its initial configuration. Modifying the content of the configuration memory corresponds to a change in the hardware implemented on the FPGA. The ICAP is a streaming interface accessible from the hardware implemented on the static part of the FPGA and for DPR purposes it receives a partial bit-stream as a continuous input stream. The hardware architecture of a system that includes the DPR feature is more complex. The use of a dedicated configuration controller to operate the ICAP interface and manage the partial bit-stream transfer is required and some border logic may be needed to isolate the reconfigurable region form the rest of the system during reconfiguration.

The time needed to perform a reconfiguration is proportional to the size of the partial bit-stream to be transferred over the ICAP, which depends on the size of the region to be reconfigured. For example, assuming the widest possible interface (32 bits) and the fastest possible clock (100 MHz) for the ICAP, the reconfiguration time of a reconfigurable region of 500 slices (which can accommodate a double-precision floating-point adder) is 300 μs [21].

The reconfiguration time is an important parameter since it introduces a time overhead of DPR that needs to be considered every time a HwA is reconfigured, leading to a computational performance loss compared to a fully static approach (not using DPR), since it increases the WCET.

3.2. The Patmos Processor and the Worst-Case Execution Timing Analysis Tool 'platin'

The processor used in this work is Patmos [2]. Patmos is a time predictable, dual-issue, RISC processor used in the T-CREST [22] multi-core platform and it has been developed specifically for use in real-time applications. The architecture of the processor is organized in a way that aims to reduce the WCET and simplify its analysis. For example, the processor pipeline is designed to avoid timing dependencies between instructions. Special WCET-optimized instruction, data caches, and local private SPMs are also used.

In real-time systems, the calculation of the WCET is fundamental to determine the system ability to respond in time. For this reason, several commercial tools and research prototypes have been developed to satisfy this need [23]. Patmos is supported by an LLVM-based compiler, also developed with a focus on WCET [24] and by the WCET analysis tool *platin* (portable LLVM-based annotation and timing analysis integration) [6]. *platin* is a comprehensive tools framework for WCET-aware compilation and WCET analysis integrated with the infrastructure of the LLVM-based compiler. Analogously to the compiler, *platin* offers dedicated support for the specific architecture of Patmos (e.g., cache modeling) and, besides other functionalities, it allows the derivation of tight WCET bounds using static methods.

The *platin* tool uses static analysis to compute the WCET of a certain code segment. This means that, in order to estimate the WCET, it only examines the software structure without code execution on real hardware. The tool works at both the bit-code level, which is the intermediate representation in LLVM, and at the machine code level. It uses the information generated and preserved during the compilation process to determine a control-flow graph annotated with flow facts provided by the user (e.g., loop iterations bound). The control-flow graph, combined with low-level timing information of the processor architecture, is therefore analyzed for the longest paths, which corresponds to a safe upper bound of the WCET of the analyzed code segment.

In this work, we use the *platin* tool in combination with reconfiguration time information provided by our *convbitstream* tool (presented later in Section 4.2) and the HwA execution time information to compute the WCET of a software task running on the Patmos processor and using reconfiguration.

4. The RT-ICAP Reconfiguration Controller

In this section, we present the reconfiguration controller RT-ICAP and the associated software tool for reconfiguration time analysis that we have developed to support time-predictable reconfiguration.

4.1. Architecture and Functionality

The reconfiguration process must be managed by a controller interfacing the FPGA configuration memory with the logic implemented on the FPGA itself. Since we target real-time systems, the time-predictability specification must also apply to the reconfiguration techniques. Most of the controllers presented in the related works offer a range of functionalities that are not strictly required by our approach to support reconfiguration, such as read-back, LUT-based reconfiguration, etc. These additional functionalities increase the hardware complexity and therefore the complexity of the hardware-level reconfiguration time

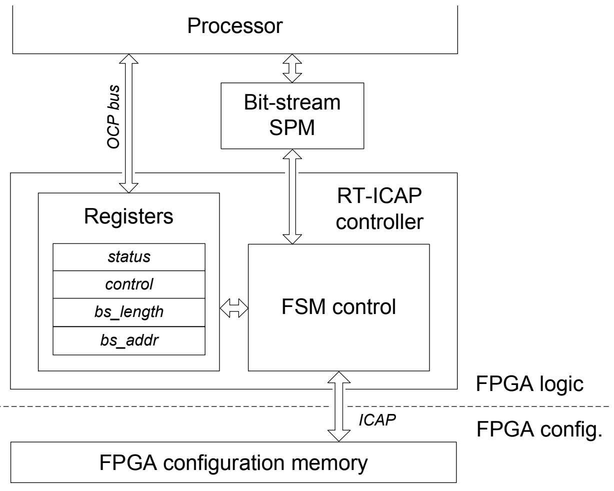


Figure 2: A block diagram of the RT-ICAP reconfiguration controller interfaced with a processor, the bit-stream SPM, and the ICAP.

analysis. Moreover, for some controllers, the detailed architecture is not public, making it impossible to perform a precise and reliable analysis.

Our RT-ICAP reconfiguration controller [25] is open-source and supports time predictable DPR in the Virtex-5, -6, and the 7-series FPGAs from Xilinx, using the ICAP interface. The controller is designed to assist processor-initiated partial reconfiguration of computation resources such as hardware accelerators in the same way as a direct memory access controller assists in moving data, making DPR and its timing analysis straightforward and easy, resulting in a small hardware implementation. Figure 2 shows a block diagram of our reconfiguration controller. The RT-ICAP controller is connected to the Patmos processor through an OCP interface [26], to a local SPM, and to the ICAP interface of the FPGA. An SPM is a relatively small private memory associated to a processor and characterized by a single clock cycle access-time. In our architecture, the SPM is used to store the reconfiguration bit-streams and it also acts as a local general-purpose memory for the processor. The fact that the SPM is not strictly dedicated to only store bit-streams is one further element that distinguishes our RT-ICAP controller from the ones discussed in the related work.

The processor controls and manages the functionality through a set of 32-bit registers. A control FSM manages the ICAP interface, the registers, and all the functionalities of the controller. The *status* register can be read by the processor to monitor the controller/reconfiguration status. The *control* register can be written by the processor to manage the controller operations. Assuming that the needed bit-streams are stored in the SPM, to initiate a reconfiguration, the processor configures the *bs_addr* register with the SPM address that points at the beginning of the bit-streams and the *bs_length* register with the length of the bit-streams. By writing the start command into the *control* register, the controller starts the reconfiguration process by autonomously fetching the bit-stream from the bit-stream-SPM and loading it into the FPGA configuration memory through the ICAP interface. The *status* register reports the controller status, including the end of the reconfiguration process.

Having the bit-streams stored in the SPM allows the controller to achieve the maximum transfer speed of the ICAP interface since there is no possible memory access bottleneck as in the case where the bit-streams must be fetched from main memory (on-chip or off-chip). The RT-ICAP can also support a bit-stream transfer directly from the processor as a sequence of writes. In this case, the processor is tasked with the copying of the bit-stream from an off-chip memory directly to the ICAP interface, and the controller only manages the transfer protocol of the ICAP interface. In this work, we use the controller only with the bit-stream placed into the local SPM.

The size of the SPM is a limiting factor since it constrains the number of bit-streams that can be locally stored and, therefore, loaded at the maximum speed of the ICAP interface. To overcome this limitation, our controller uses lossless run-length encoding (RLE) compression techniques to decrease the size of the partial bit-streams. The bit-streams are compressed by the software tool associated with the RT-ICAP (presented in the following subsection) and it is decompressed in hardware by the controller. We opted to use a simple RLE compression instead of more advanced techniques (such as Huffman, Arithmetic, Lempel-Ziv, etc. [27, 28, 29]), due to its simplicity, which results in a small hardware implementation for decompression and easy timing analysis. The main idea used in the RLE compression technique is to store sequences of repeated characters (called 'data run') as a single character and a character occurrence count. Therefore, in the compressed bit-streams, a data run appears as an escape value to signal the beginning of a compressed sequence, followed by the count and the data itself. Our RLE technique is used in addition to the frame-based compression offered by the Xilinx tools. A frame is the configuration memory segment correspondent to the smaller allowed reconfigurable region, and its size is in the order of KB. Since the two techniques work at different levels of data granularity, our RLE compression can further increase the compression ratio obtained using the Xilinx native compression [25].

4.2. Tool Support and Timing Analysis

The RT-ICAP controller is supported by a software tool, named *convbitstream*. It compresses the bit-streams, converts them to the format required by the RT-ICAP controller and, most importantly, it computes the reconfiguration time the controller takes to perform the reconfiguration for each compressed bit-streams. The reconfiguration time is from the moment the processor initiates the reconfiguration and until the partial bit-stream is entirely written into the FPGA's configuration memory and the reconfigurable region is ready to be used. This time interval is needed to perform WCET analysis at application-level when using the reconfiguration feature.

The reconfiguration time depends on the properties of the RT-ICAP controller architecture and on the properties of the compressed bit-streams, it can be computed using Equation (1). It consists of the clock period T_{clk} multiplied by a number of clock cycles as a sum of three contributions: the cycles needed to initialize and finalize a reconfiguration, the cycles required to transfer the compressed bit-streams into the RT-ICAP controller, and the additional cycles needed to expand compressed data runs and write these into the configuration memory of the FPGA.

$$T_{rec} = T_{clk}\{n_{oh} + n_s + \sum_{i=1}^{n_r} (R_{i\_len} - 1)\} \tag{1}$$

More specifically, n_{oh} is the overhead required by the RT-ICAP controller for starting and finishing a reconfiguration. n_s is the amount of words in the compressed bit-streams, which includes the number of simple uncompressed characters, the number of compressed data runs, and the number of escape sequences. Writing a word into the ICAP interface takes one clock cycle. The third contribution is due to decompression and it accounts for the number of additional clock cycles required for writing repeating characters into the ICAP interface. Here, R_{i_len} is the number of times a compressed character repeats in the i-th data run and n_r is the amount of data runs. During these decompression intervals the interface towards the SPM stalls.

5. Experimental Setup

This section describes the experimental setup used to produce the results. More specifically, we describe the hardware platform and the set of HwAs from the TACLe benchmark suite that we use [4].

5.1. Hardware Platform

The hardware platform used in our experiments is shown in Figure 3. The I/O devices are connected to the Patmos processor using a bus that implements a subset of the open-core protocol (OCP) [26]. The HwA is connected to a shared memory (HwA-SPM) for data exchange with Patmos and to a controller (HwA-ctrl) used to manage the HwA and provide the current status to the processor. For communication with the HwA-ctrl, the HwA uses the *ap_ctrl_hs* interface protocol, as defined in [5, p. 89].

The HwA-SPM is divided into a certain number of banks decided at synthesis time. The HwA-SPM appears as a single address space to the processor, but the banks can be accessed in parallel by the HwA to increase the memory bandwidth towards it. Note that the HwA-SPM is not the local SPM of Patmos, which is used by the processor to store easily accessible data and instructions.

Both the HwA and the HwA-SPM reside in the reconfigurable region, since the number of memory banks used is specific for the HwA and, therefore, needs to be reconfigured together with the HwA.

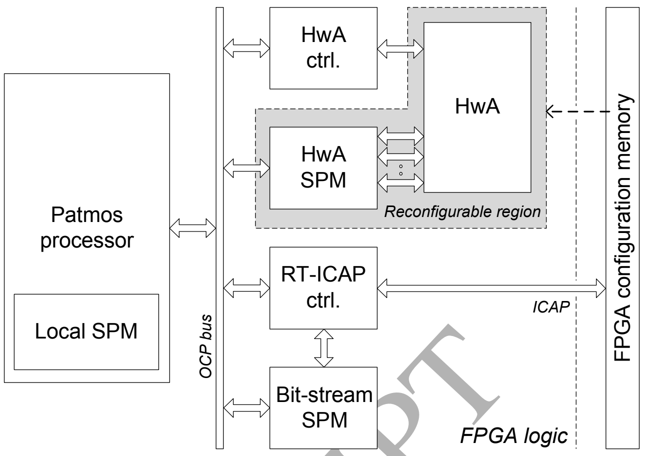


Figure 3: An overview of the hardware platform used for the experiments. All the I/O devices are connected to the processor using the OCP bus.

The reconfiguration is managed by the RT-ICAP controller presented in Section 4. The RT-ICAP controller can modify the content of the FPGA configuration memory through the ICAP interface. Therefore, by writing a partial bit-stream (stored in the bit-stream SPM) into this memory, the content of the reconfigurable region is dynamically modified. The dashed arrow in the upper right corner of Figure 3 characterizes this dependency.

Assuming that the bit-streams are available in the bit-stream SPM, the full operational flow of the system to use the HwA is as follows:

1. Patmos requests the RT-ICAP controller to reconfigure the needed HwA;
2. Patmos moves the data to be processed into the HwA SPM;
3. Patmos activates the HwA by interacting with the HwA controller;
4. When the HwA has finished, Patmos can read the processed data from the HwA SPM.

Step 1 can be skipped if the reconfigurable area does not need to be reconfigured. During step 3 when the HwA is running, the processor is free to execute other operations.

5.2. Benchmarks and Hardware Accelerators

The HwAs used in this paper are based on code from four benchmarks from the TACLe suite [4], which is a collection of open-source C programs, for timing analysis and real-time related research. For the selected benchmarks, the computationally intensive part of the program is identified and moved into hardware. The benchmark is then modified to interact with the HwA to perform the section of the program that was moved into hardware. When DPR is used, the reconfiguration of the dynamic region with the needed HwA is performed before using the HwA.

The HwAs used in this paper are generated using Xilinx Vivado HLS, which is an automated design process able to transform high-level language code, such as C, into functionally equivalent synthesizable RTL HDL code [5]. In our case, it is used to transform C code from the TACLe real-time benchmarks into VHDL.

The benchmarks have been chosen to be representative of HwAs working on large data sets, small data sets, and data streams. The data type used in all the benchmarks, where not differently specified, is single-precision floating-point. In the following, we list and provide a brief description of the functionality of each benchmark and the characteristics of the associated HwA:

- **Matrix multiplication**: This benchmark executes the matrix multiplication between two square matrices. For this benchmark, we have generated different specialized HwAs for matrices of size 4×4, 16×16, and 32×32. Moreover, we have generated a generic HwA for matrix multiplication which can take any given matrix size up to 32×32. This is used to analyze how the specialized HwAs combined with partial reconfiguration perform in comparison with a generic HwA.
- **Matrix inversion**: This benchmark computes the matrix inversion operation on a square matrix. For this benchmark, we have also generated HwAs for matrices of size 4×4, 16×16, and 32×32.
- **2D FIR filter**: This benchmark performs a bidimensional FIR filtering on a matrix of size M×N using a 3×3 coefficient mask. This kind of filtering is commonly used for smoothing or sharpening bidimensional data sets. More specifically, the benchmark computes the cross-correlation between a 4×4 area surrounding each value and the coefficient mask. Zero-padding is performed at the matrix edges to satisfy the filter conditions. For this benchmark, we have generated a HwA for a matrix of size 3×3, corresponding to the minimum input matrix size.
- **Filterbank**: This benchmark implements a filterbank with FIR filters for multi-rate signal processing. The input signal is passed into eight different FIR filters. The filtered signals are then down-sampled and up-sampled again. The up-sampled signal is passed through a second set of FIR filters and finally the outputs are summed together. Typically, some data processing is performed between the down-sampling and the up-sampling. In our benchmark, we do not perform any processing. For this benchmark, we have generated a HwA where the input data and the two sets of filter coefficients are passed as arguments.

Table 1: Hardware resources characterization of our RT-ICAP controller compared with other designs presented in related works.

Controller	Target FPGA	Hardware resources		
		FF	LUT	BRAM
RT-ICAP	Artix-7	105	289	0
PRC [18]	Kintex-7	1 270	1 075	0
ZyCAP [16]	Zynq-7000	806	620	0
DPRM [19]	Virtex-6	77	109	0
D^2PR [20]	Virtex-6	112	249	0
XPS_HWICAP [15]	Virtex-5	745	741	3

6. Results and Discussion

This section presents the experimental evaluation of the use of DPR in real-time systems. At first, we present the hardware characterization of our RT-ICAP controller. This is followed by the results obtained with the use of the benchmarks in terms of reconfiguration overhead, hardware utilization, bit-stream size and compression, comparison with a pure software solution, and trade-offs between specialized and generic HwAs.

All the results presented in this section are produced using Xilinx Vivado and HLS (v16.4) and targeting the Xilinx Artix-7 FPGA (model XC7A100T-1CSG324C). The size of the reconfigurable region used in all the experiments is 1500 slices, which contains 6000 look-up tables (LUTs), 12000 flip-flops (FFs), 30 block-RAMs (BRAMs), and 40 digital signal processing (DSP) elements. All the synthesis and implementation properties of the tools were set to their defaults. For Xilinx HLS, optimization directives have been used aiming to balance the area vs. performance trade-off of the generated accelerators. The data size used for the ICAP interface is 32 bits.

6.1. RT-ICAP Hardware Characterization

Before evaluating the potential benefits deriving from the use of DPR in real-time systems, we present an evaluation of our RT-ICAP controller. Table 1 presents the hardware resource utilization for our controller and the other controllers introduced in the related works, in terms of FFs, LUTs, and BRAMs. For these controllers, the results presented in the table are retrieved from the respective publications. The second column of Table 1 specifies the target FPGA used to produce the results. As previously mentioned, for our controller, the results are produced using the Xilinx Artix-7 FPGA. Even if the results for the other controllers are produced targeting different FPGAs, they are quantitatively comparable, since all FPGAs use 6-input LUTs.

From the results, we can observe that our controller is comparable in size to the controller D^2PR [20] (synchronous version without error check) and relatively bigger than the controller DPRM [19]. Among the different controllers,

Table 2: Contributions to the WCET in clock-cycles and the number of times ($N_{95\,\%}$) the HwA has to be used to reach a performance that is 95 % of the one of a system that uses a static HwA.

Benchmark	Software (C_{sw})	Hardware (C_{hwa})	Reconfig. (T_{rec})	$N_{95\,\%}$
Matrix mult.				
- 4×4	3 203	42	133 436	781
- 16×16	30 345	1 107	130 786	79
- 32×32	114 816	8 351	133 823	21
Matrix inv.				
- 4×4	2 307	793	130 772	802
- 16×16	21 363	12 168	131 885	75
- 32×32	78 859	55 223	130 071	18
2D FIR filter	3 174	137	132 098	758
Filterbank	46 450	216 264	132 152	10

these two are the most similar ones to our RT-ICAP controller, in terms of the architecture. However, they only offer simple transfer of bit-streams from a memory to the ICAP interface. Our controller offers additional functionalities, such as support of bit-stream decompression, and a register-based (*status*/*control*) interface. With regards to the other controllers listed in Table 1, our RT-ICAP is considerably smaller. This is due to the fact that it does not implement those functionalities that are not time-predictable or not needed by our reconfiguration approach, such as bit-stream read-back.

6.2. Reconfiguration Overhead

The first set of results is related to the WCET of the benchmarks and aims to characterize the overhead of reconfiguration over actual computation time.

Three factors contribute to the WCET of a benchmark: C_{sw}, C_{hwa}, and T_{rec}. The first contribution C_{sw} is the WCET of the software section of the benchmark produced by the *platin* time-analysis tool. This includes the WCET of the data transfer to and from the HwA-SPM and the WCET of the setup of the HwA. The second contribution C_{hwa} is the time needed by the HwA to perform the computation in the worst-case. This result is produced by the Vivado HLS tool. The third contribution T_{rec} is the reconfiguration time needed to perform the reconfiguration of the reconfiguration region. This time interval is produced by the *convbitstream* tool using Equation 1. All time periods are measured in clock cycles.

Table 2 presents the values of these three contributions for all the benchmarks considered in this work. The total WCET of a benchmark that uses DPR is denoted C_{tot_dpr} and it can be calculated using equation (2), where N is the number of computations executed by the HwA after a reconfiguration.

$$C_{tot\_dpr} = T_{rec} + N\,(C_{sw} + C_{hwa}) \tag{2}$$

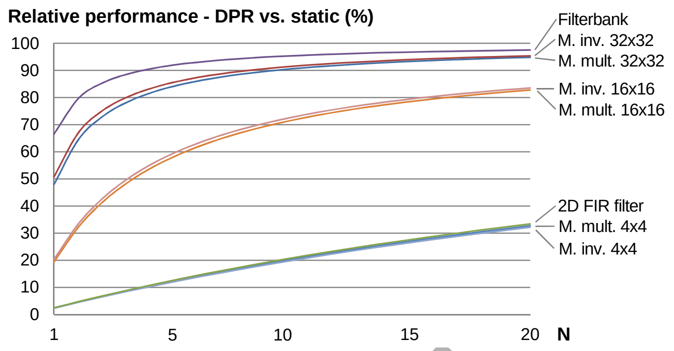


Figure 4: Plot showing the performance when using DPR relative to the performance when using static HwAs for values of $N \in [1, 20]$.

The reconfiguration time, T_{rec}, represents an overhead, and the effect of this overhead is reduced with the number of times the HwA is used. The last column of Table 2 shows the number of times the HwA has to be used in order to reach a performance that is 95 % of the performance of a system that uses a static HwA. To obtain this values, the same accelerators have been used for both the reconfigurable solution and the static one. It is possible to observe that, for the HwAs that perform computationally intensive tasks, such as *Matrix multiplication* and *Matrix inversion* for 32×32 matrices and *Filterbank*, the value of $N_{95\,\%}$ is very low. Low values of $N_{95\,\%}$ show that the reconfigurable solution may be particularly beneficial, even if only a small number of computations are required, since the loss of performance is compensated for by lower hardware cost as discussed in the next subsection.

Figure 4 provides additional insight into the relation between reconfiguration overhead and the number of times, N, that the HwA is used. The figure shows the performance when using DPR relative to the performance when using static HwAs for values of $N \in [1, 20]$. In the figure, it is possible to observe how results for similar benchmarks tend to cluster together. In the top of the plot, we can find the curves related to *Filterbank* and the benchmarks operating on 32×32 matrices. Right below, in the second group, we can find the curves related to the benchmarks operating on 16×16 matrices. For these two groups, the solution using DPR becomes comparable to the static approach, in terms of computational performance, for low values of N. Finally, the curves related to the *2D FIR filter* and to the benchmarks operating on 4×4 matrices are located in the lower part of the plot, showing that it is unlikely that a real application can benefit from using DPR.

6.3. Hardware Utilization

One of the advantages of using DPR is the possibility of reducing the hardware utilization in the case when some of the hardware resources implemented in a system are only utilized for a limited amount of time, since the HwAs can be loaded in the reconfigurable regions when needed.

Table 3 shows the hardware utilization results for the main components of the hardware platform (shown in Fig-

Table 3: Hardware utilization results for the hardware platform and for all the HwAs used in the experiments.

Entity	LUT	FF	BRAM	DSP
Patmos	4 931	3 602	8.5	4
HwA controller	7	4	0	0
ICAP controller	289	105	0	0
Matrix mult.				
- 4×4	1 270	1 138	0	20
- 16×16	1 979	2 523	0	20
- 32×32	2 763	4 048	0	20
Matrix inv.				
- 4×4	2 051	2 017	0.5	5
- 16×16	3 425	3 725	0.5	20
- 32×32	4 080	4 636	0.5	20
2D FIR filter	1 816	1 987	0	10
Filterbank	5 126	7 411	3	10
Generic matrix mult.	2 912	4 037	0	5

ure 3) and for all the HwAs used in the experiments, in LUTs, FFs, BRAMs, and DSP elements.

Considering a hypothetical application that includes all the functionality of the benchmarks used in this work, it is possible to observe that the minimum size of the reconfigurable region should be enough to contain the largest hardware requirements across all the HwAs. In our case, this corresponds to 5 126 LUTS, 7 411 FFs and 3 BRAMs to fit the *Filterbank* HwA, and 20 DSP elements to fit the *Matrix multiplication* and *Matrix inversion* HwAs requirements. In a static solution, where all the HwAs need to be implemented, the total hardware cost would be roughly equal to the sum of the hardware resources of each HwA. This leads to a relevant saving in the hardware cost since the required minimum size of the reconfigurable region is the 23 % of LUTs and the 27 % of FFs of the estimated hardware cost of a static solution.

Taking the hardware resource utilization results and the performance results presented in the previous subsection into account, we can observe that, for a value of N sufficiently high, DPR leads to a more efficient use of FPGA resources, while maintaining comparable computational performance.

6.4. Bit-Stream Size and Compression Ratio

As previously mentioned, the size of the bit-stream SPM limits the number of bit-streams that can be locally stored. The *convbitstream* controller performs an RLE compression on the raw bit-stream to maximize this number. Table 4 reports the size, the reconfigurable region utilization, and the compression ratio of the partial bit-streams for all the benchmarks. The bit-stream size shown in the table refers to the bit-stream compressed with both the Xilinx native compression and our RLE compression. The utilization

Table 4: Size, reconfigurable region utilization, and compression ratio of the bit-streams for all the benchmark. The compression ratio accounts only for the RLE compression performed by the *convbitstream* tool.

Benchmark	Bit-stream size (KB)	Rec. region utilization	Compression ratio
Matrix mult.			
- 4×4	209.3	35 %	2.3
- 16×16	274.3	62 %	1.7
- 32×32	328.1	83 %	1.5
Matrix inv.			
- 4×4	217.9	49 %	2.2
- 16×16	336.1	86 %	1.4
- 32×32	354.5	94 %	1.3
2D FIR filter	233.8	49 %	2.1
Filterbank	369.8	100 %	1.3

is defined as the percentage of the reconfigurable region, in term of slices, used by the hardware accelerator implemented in it. The compression ratio is defined as the size of the uncompressed bit-stream divided the by the size of the compressed one, and it only accounts for our RLE compression performed on bit-stream already compressed with Xilinx native compression.

From the results, we can observe that the RLE compression produces bit-streams ranging from 1.3 to 2.3 times smaller than the original size, with an average compression ratio of 1.7. This leads to a memory saving of 1.5 MB over a total size of 3.7 MB needed for storing all uncompressed bit-streams. As expected, the compression delivers a higher ratio for those accelerators that have lower utilization of the reconfigurable region, since these bit-streams contains long runs of zeros for the unused logic. Moreover, it is very interesting to observe that bit-streams already compressed with the Xilinx native compression can be further compressed using RLE, due to the fact that the two compression techniques work at different levels of data granularity. Xilinx compression works at a level of frames, while the RLE compression operates at a level of 32-bit words.

6.5. Comparison with Software

In this set of experiments, we evaluate the speed-up in terms of WCET between the solution that uses hardware accelerators in combination with DPR and a pure software solution. As previously explained, using a reconfigurable solution instead of a static one introduces an overhead that decreases with the number of computations N executed by the hardware accelerator after a reconfiguration. For this set of experiment, we evaluate the speed-up assuming $N = N_{95\%}$ (reported in the last column of Table 2). Table 5 presents the WCET and the speed-up results for all the benchmarks considered in this work. All WCET listed in the table are normalized with respect to N in order to show the WCET time of a single execution. However, the speed-ups are calculated as ratios between the total WCETs for $N = N_{95\%}$.

The Patmos processor is not equipped with a floating-point unit. Therefore, to execute floating-point operations, it utilizes equivalent software functions from the LLVM project [30]. Comparing a solution that uses floating-point accelerators (matrix multiplier, matrix inverter, etc.) against software running on a processor that is not equipped with a floating-point unit can produce biased results, since the speed-up does not characterize the ability of the hardware accelerator to exploit parallelism, but mainly the speed-up related to faster floating-point operations. To have a fairer comparison, in addition to the results where the floating-point operations are executed in software, we also present results for the software benchmark that uses 32-bit integer data instead of single-precision floating-point, since these operations are supported in hardware by the processor. For the pure software solution, the data to be processed is placed in the local SPM as it is done in the accelerator-based solution. This avoids data cache misses to be accounted into the WCET.

The first column of the table reports the WCET of the reconfigurable solution C_{tot_dpr} computed using the values presented in Table 2. The second and third columns present the WCET of the pure software solution obtained with the WCET analysis tool *platin* for floating-point C_{sw_fp} and integer data types C_{sw_int}, respectively. Finally, the last two columns show the speed-up calculated as the ratio between the WCET of the solution that uses hardware accelerators in combination with DPR and a pure software solution

From the results, we can observe that the speed-up results for the software using floating-point data type are very high. As previously mentioned, this can be explained by the fact that Patmos is not equipped with a floating-point unit. The speed-up results for the software using integer data type are more realistic and give a concrete grasp of the value that can be obtained by using reconfigurable accelerators, showing speed-ups for the WCET from 1.2 to 4.1. In one case, for the 4×4 matrix multiplication, the speed-up is less than 1. This means that the reconfigurable accelerator solution performs worse than the software one. This can be explained by the fact that the execution time of the benchmark is dominated by data transfer and the acceleration provided by using hardware is not enough to compensate for the reconfiguration overhead. In contrast, the execution time of the *Filterbank* benchmark is dominated by computation with little data transfer, leading to a speed-up of one order of magnitude higher than the other benchmarks.

It should be noted that the results presented in Table 2 are obtained using a soft-processor implemented on an FPGA. This is the Patmos processor used in the scope of this work. In general, if a hard-processor is used, we can expect a reduction of the speed-up results presented above.

6.6. Specialized vs. Generic Accelerator

The goal of this experiment is to determine what benefits a very specific HwA combined with DPR may bring compared to a generic one, since the reconfiguration feature can be used just to change the type of specialized HwA based on current requirements.

As previously mentioned, for *Matrix multiplication* we have generated specialized HwAs for matrix sizes of 4×4, 16×16, 32×32, and a generic HwA which can take in input matrices of any size up to 32×32, which includes additional logic and requires additional computation time for managing and setting-up the computation for different matrix sizes. The idea is to investigate the trade-offs between using DPR to switch between multiple specialized HwAs and using a more general HwA, in terms of WCET and hardware utilization.

Table 6 shows the WCETs for both the specialized HwAs and the generic one in clock cycles for the *Matrix multiplication* benchmark for the three different matrix sizes. Table 6 also shows the reconfiguration time for the specialized HwA.

Contrarily to the experiment presented in Subsection 6.2, where the solution using reconfiguration is always slower than the static one, in this experiment the solution using reconfiguration can be faster than a static solution since the specialized HwAs are faster than the generic ones.

Therefore, after a certain amount of computation, the overhead introduced by the reconfiguration time will be compensated for by the difference in speed between the specialized and the general HwA. The value $N_{100\,\%}$, shown in the last row of Table 6, is the threshold value of N (number of times the HwA is used) in which the general and the specialized HwAs, including reconfiguration, are equivalent in performance. For values of $N > N_{100\,\%}$, the use of the specialized HwA combined with reconfiguration outperform the general HwA.

In terms of hardware, it is possible to observe from Table 3 that the minimum size of the reconfigurable region should be enough to contain the specialized *Matrix multiplication* HwA for size 32×32, which is smaller than the hardware resources needed to implement the generic HwA (last row of Table 3).

7. Conclusion

In this paper, we have experimentally evaluated the potential advantages from utilizing hardware accelerators in combination with the DPR feature of FPGAs, using four representative test cases derived from the real-time TACLe benchmark suite. We have also presented our hardware/software infrastructure supporting time-predictable reconfiguration, consisting of the RT-ICAP controller and the associated software tools *convbitstream*.

The experiments have shown that in the case where an application performs a sequence of tasks in a way that allows different HwAs to be loaded into the reconfigurable

Table 5: WCET and relative speed-up for the solution that uses hardware accelerators in combination with DPR and pure software solutions using floating-point and integer data types.

Benchmark	HwA + DPR (C_{tot_dpr})	Software (float.) (C_{sw_fp})	Software (int.) (C_{sw_int})	Speed-up (float.)	Speed-up (int.)
Matrix mult.					
- 4×4	3 416	208 451	1 725	61.0	0.5
- 16×16	33 108	13 072 787	71 493	394.9	2.2
- 32×32	129 540	104 301 204	530 534	805.2	4.1
Matrix inv.					
- 4×4	3 263	280 084	48 894	85.8	15.0
- 16×16	35 289	14 607 172	497 190	413.9	15.1
- 32×32	141 308	113 617 924	2 169 670	804.0	15.4
2D FIR filter	3 485	471 460	4 155	135.3	1.2
Filterbank	275 929	864 106 383	15 279 271	3 131.6	55.4

Table 6: WCET results for general and specialized HwAs, reconfiguration time for the specialized HwAs, and the minimum number of operations $N_{100\,\%}$ for which it is convenient to use DPR.

	4×4	16×16	32×32
General HwA	8 028	49 438	152 933
Specialized HwA	3 245	31 452	123 167
Reconfig. time	133 436	130 786	133 823
$N_{100\,\%}$	28	8	5

region when needed, the use of DPR can lead to a significant reduction in the hardware cost if the tasks moved into hardware are sufficiently computationally intensive. In our benchmarks, this was observed for computationally intensive tasks, such as the *Filterbank* and the benchmarks operating on 32×32 matrices. In comparison with a pure software solution, the reconfigurable solution delivers speed-ups, in term of WCET, ranging from 1.2 to 4.1. The experiments performed to analyze the trade-offs between specialized and general HwAs suggested that, for computationally intensive tasks, the use of the specialized HwAs combined with reconfiguration is advantageous with respect to the general HwA, both in terms of performance and hardware utilization.

Finally, we have also shown by our RT-ICAP controller is comparable or smaller in size than others controller developed in academia or industry and the supported RLE compression can substantially reduce the bit-streams memory footprint, producing bit-streams from 1.3 to 2.3 times smaller than the original size.

Source Access

The source code used for synthesis is available at `https://github.com/A-T-Kristensen/patmos_HLS/tree/master/hls` and the code required to run on the T-CREST platform is available at `https://github.com/A-T-Kristensen/patmos/tree/patmos_hls`. The RT-ICAP controller and the *convbitstream* tool is available at `https://github.com/t-crest/reconfig`. The full T-CREST platform is available at `https://github.com/t-crest/`. The entire work is open-source under the terms of the simplified BSD license.

Bibliography

References

[1] XILINX, "UG909: Vivado Design Suite User Guide - Partial Reconfiguration (v2017.1)," Tech. Rep., 2017, online (last accessed: Feb. 2018).

[2] M. Schoeberl, P. Schleuniger, W. Puffitsch, F. Brandner, C. W. Probst, S. Karlsson, and T. Thorn, "Towards a time-predictable dual-issue microprocessor: The Patmos approach," in *Proc. of the Workshop on Bringing Theory to Practice: Predictability and Performance in Embedded Systems (PPES)*, 2011, pp. 11–20.

[3] M. Schoeberl, W. Puffitsch, S. Hepp, B. Huber, and D. Prokesch, "Patmos: A time-predictable microprocessor," *Real-Time Systems*, 2018, accepted for publication.

[4] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. B. Sørensen, P. Waegemann, and S. Wegener, "TACLeBench: A benchmark collection to support worst-case execution time research," in *Proc. of the 16th International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2016, p. 10.

[5] XILINX, "UG902: Vivado Design Suite User Guide - High-Level Synthesis (v2017.1)," Tech. Rep., 2017, online (last accessed: Feb. 2018).

[6] S. Hepp, B. Huber, J. Knoop, D. Prokesch, and P. P. Puschner, "The platin tool kit - The T-CREST approach for compiler and WCET integration," in *Proc. of the 18th Kolloquium Programmiersprachen und Grundlagen der Programmierung (KPS)*, 2015.

[7] L. Pezzarossa, A. T. Kristensen, M. Schoeberl, and J. Sparsø, "Can real-time systems benefit from dynamic partial reconfiguration?" in *Proc. of the 3rd Nordic Circuits and Systems Conference (NORCAS)*. IEEE, Oct 2017, pp. 1–6.

[8] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171–210, 2002.

[9] R. Tessier, K. Pocek, and A. DeHon, "Reconfigurable computing architectures," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 332–354, 2015.

[10] K. Wu, "Reconfigurable architectures: from physical implementation to dynamic behavoir modelling," Ph.D. dissertation, Dept. of Informatics and Mathematical Modelling, Technical University of Denmark, 2007.

[11] D. Koch, C. Beckhoff, and J. Teich, "ReCoBus-Builder - a novel tool and technique to build statically and dynamically reconfigurable systems for FPGAs," in *Proc. of International Conference on Field Programmable Logic and Applications.* IEEE, 2008, pp. 4 629 918, 119–124.

[12] S. Banerjee, E. Bozorgzadeh, and N. D. Dutt, "Integrating physical constraints in HW-SW partitioning for architectures with partial dynamic reconfiguration," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 11, pp. 1189–1202, Nov 2006.

[13] R. Cattaneo, R. Bellini, G. Durelli, C. Pilato, M. D. Santambrogio, and D. Sciuto, "Para-sched: A reconfiguration-aware scheduler for reconfigurable architectures," in *Proc. of the Parallel Distributed Processing Symposium Workshops, IEEE*, May 2014, pp. 243–250.

[14] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo, "A framework for supporting real-time applications on dynamic reconfigurable FPGAs," in *Proc. of the Real-Time Systems Symposium (RTSS).* IEEE, Nov 2016, pp. 1–12.

[15] XILINX, "DS586: LogiCORE IP XPS HWICAP product specifications (v5.01a)," Tech. Rep., 2011, online (last accessed: Feb. 2018).

[16] K. Vipin and S. A. Fahmy, "ZyCAP: Efficient partial reconfiguration management on the Xilinx Zynq," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 41–44, Sept 2014.

[17] XILINX, "UG761: LogiCORE IP XPS HWICAP product specifications (v13.1)," Tech. Rep., 2011, online (last accessed: Feb. 2018).

[18] ——, "PG139: LogiCORE IP PRC product guide (v1.0)," Tech. Rep., 2015, online (last accessed: Feb. 2018).

[19] J. Tarrillo, F. A. Escobar, F. L. Kastensmidt, and C. Valderrama, "Dynamic partial reconfiguration manager," in *Proc. of the Latin American Symposium on Circuits and Systems (LASCAS).* IEEE, 2014, pp. 1–4.

[20] S. D. Carlo, P. Prinetto, P. Trotta, and J. Andersson, "A portable open-source controller for safe dynamic partial reconfiguration on Xilinx FPGAs," in *Proc. of the 25th International Conference on Field Programmable Logic and Applications (FPL)*, 2015, pp. 1–4.

[21] L. Pezzarossa, M. Schoeberl, and J. Sparsø, "Reconfiguration in FPGA-based multi-core platforms for hard real-time applications," in *Proc. of the International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC).* IEEE, 2016, pp. 1–8.

[22] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, S. Hepp, B. Huber, A. Jordan, E. Kasapaki, J. Knoop, Y. Li, D. Prokesch, W. Puffitsch, P. Puschner, A. Rocha, C. Silva, J. Sparsø, and A. Tocchi, "T-CREST: Time-predictable multicore architecture for embedded systems," *Journal of Systems Architecture*, vol. 61, no. 9, pp. 449–471, 2015.

[23] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution time problem – overview of methods and survey of tools," *Trans. on Embedded Computing Systems*, vol. 7, no. 3, pp. 1–53, 2008.

[24] P. Puschner, R. Kirner, B. Huber, and D. Prokesch, "Compiling for time predictability," in *Computer Safety, Reliability, and Security*, ser. Lecture Notes in Computer Science. Springer, 2012, vol. 7613, pp. 382–391.

[25] L. Pezzarossa, M. Schoeberl, and J. Sparsø, "A controller for dynamic partial reconfiguration in FPGA-based real-time systems," in *Proc. of the 20th International Symposium on Real-Time Distributed Computing (ISORC).* IEEE, 2017, pp. 92–100.

[26] OCP official website, "Webpage: http://www.accellera.org/downloads/standards/ocp/," (last accessed: Feb. 2018).

[27] S. Hauck and W. D. Wilson, "Runlength compression techniques for FPGA configurations," in *Proc. of the Annual Ieee Symposium on Field-programmable Custom Computing Machines (FCCM).* IEEE Computer Society, 1999, pp. 286–7, 286–287.

[28] Z. Li and S. Hauck, "Configuration compression for Virtex FPGAs," in *Proc. of the Annual Ieee Symposium on Field-programmable Custom Computing Machines (FCCM).* Institute of Electrical and Electronics Engineers Inc., 2001, pp. 147–159.

[29] D. Koch, C. Beckhoff, and J. Teich, "Hardware decompression techniques for FPGA-based embedded systems," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 2, no. 2, pp. 9:1–9:23, Jun. 2009.

[30] C. Lattner and V. Adve, "The LLVM instruction set and compilation strategy," CS Dept., Univ. of Illinois at Urbana-Champaign, Tech. Rep. UIUCDCS-R-2002-2292, Aug 2002.

Biography of all authors

Luca Pezzarossa received his MSc in Electronic Engineering from the Marche Polytechnic University (Italy) in 2014. Currently, he is a PhD student at the Technical University of Denmark where he is working with reconfigurable time-predictable architectures. His research interests include reconfigurable computing, real-time systems, embedded applications, networks-on-chip, and system-on-chip design.

Andreas Toftegaard Kristensen received his BSc in Electrical Engineering from the Technical University of Denmark in 2017. Currently, he is a MSc student at the same university where he is focusing his studies on hardware acceleration for machine learning applications. His study and research interests include hardware acceleration, high-level synthesis, machine learning, and digital arithmetic.

Martin Schoeberl received his PhD from the Vienna University of Technology in 2005. From 2005 to 2010 he has been Assistant Professor at the Institute of Computer Engineering. He is now Associate Professor at the Technical University of Denmark. His research interest is on hard real-time systems, time-predictable computer architecture, and real-time Java. Martin Schoeberl has been involved in a number of national and international research projects: JEOPARD, CJ4ES, T-CREST, RTEMP, and the TACLe COST action. He has been the technical lead of the EC funded project T-CREST. He has more than 100 publications in peer reviewed journals, conferences, and books.

Jens Sparsø is professor at the Technical University of Denmark (DTU). His research interests include: design of digital circuits and systems, design of asynchronous circuits, low-power design techniques, application-specific computing structures, computer organization, multi-core processors, and networks-on-chips – in short hardware platforms for embedded and cyber-physical systems. He has published more than 90 refereed conference and journal papers and is coauthor of the book "Principles of Asynchronous Circuit Design - A Systems Perspective" (Kluwer, 2001), which has become the standard textbook on the topic. He received the Radio-Parts Award and the Reinholdt W. Jorck Award in 1992 and 2003, in recognition of his research on integrated circuits and systems. He received the best paper award at ASYNC 2005, and one of his papers was selected as one of the 30 most influential papers of 10 years of the DATE conference.

Luca Pezzarossa

Andreas Toftegaard Kristensen

Martin Schoeberl

Jens Sparsø